1993

XBuild: Flexible, generic, X-based user interface tools

Marcus Cannava

Flexible, Generic, X-Based User Interface Tools

Marcus N. Cannava

May 17, 1993

Rochester Institute of Technology
College of Applied Science and Technology
Department of Computer Science

XBUILD
Flexible, Generic, X-Based User Interface Tools

CERTIFICATE OF APPROVAL

This Project is Submitted in Partial Fulfillment
of the Requirements for a Degree of

Master of Science
in Computer Science

by Marcus N. Cannava

Approved By:

Project Advisor: _____
Dr. Peter G. Anderson
Computer Science Department

Reading Member: _____
Daryl Johnson
Computer Science Department

Reading Member: _____
Dr. Jim Heliotis
Computer Science Department

Contents

Dedicated to my wife, Ruth Cannava, without whom this project could never have been completed. I love you!

Chapter 1

Introduction to the Package

1.1 What is XBUILD?

XBUILD is a generic form builder. There are many applications that require the use of a customized, electronic form to gather information from a group of users. Many interface builders have been written to aid programmers in creating friendly front-ends to their programs, but few interface builders have been targeted for the non-programmer to create forms that will run on their own, without having to write any code to actually utilize the interface. To this end, XBUILD provides a set of tools that help create self-running electronic forms which can gather data, access data files from other programs, or act as front-ends to programs that have no friendly interface.

XBUILD has a flexible scripting language with twenty-four commands that allow you to tailor the forms you build to suit your needs. Fields can have custom presets to allow or disallow certain types of input, and can automatically format the data to whatever form you desire.

XBUILD is divided into three parts. The primary developer's tool is XBuild, which is used to graphically build forms and save them into a file. The end-user's tool is XRead, which reads the files XBuild generates, displays forms to the user, and executes the commands in the form's script. There is also a database manager, XStat, which helps the developer manage the data collected. It also generates reports in whatever format you wish.

1.2 Background

This project was written using the MIT X11R5 release of the X window system. My first choice was whether or not to implement the project using Xlib, the lowest layer of X programming, or use one of the many X toolkits available.

To see how feasable it was, I built a simple prototype of the program using the basic Xlib calls (a copy of this prototype program is included in the project package) and writing the support routines myself. I quickly discovered that this was an impractical option, as I would have had to write most of a toolkit myself just to get my project off the ground. I reasoned that using an X toolkit would simply be saving me the basic work of implementing widgets, mouse event handlers, and all the other support functions that make up a good X toolkit.

But now, which toolkit to use? Among my choices were Motif, OpenLook (XView), or the basic X Toolkit (Xt) with the Athena widget set. My decision was made easier by the fact that we do not have Motif, leaving simply OpenLook or Athena[1]. I compared the Athena and OpenLook feature sets, and decided that while OpenLook was certainly more complete, Athena implemented all that I needed to use and was simpler[2]. I made my decision to go with the Athena widget set.

Xt met all of my needs. I no longer had to write code to interpret raw X events into menu items, among other things. The Athena widget set, on the other hand, left a few things to be desired. Between the features that Athena didn't have and the bugs that exist in this release, I found myself writing code to work around Athena that I'm sure wouldn't have been necessary in either Motif or OpenLook.

For example, a serious bug exists that will not let you change a widget's border width once it has been created. It can be specified when `XtVaCreateManagedWidget()` is called, but never changed after this (even though the return code from `XtVaSetValues()` indicates success). Another bug exists that restricts widget repositioning when it is the child of a "Form"

[1] I would probably have used Motif, if it had been available. I have plans to re-write the XBUILD package using Motif.

[2] Also, I am personally not fond of OpenLook. As if in support of my decision not to use OpenLook, a recent announcement in [2] says that Sun has decided to include Motif support in future revisions of Solaris, eventually replacing OpenLook.

widget. I spent a lot of time trying to debug these sections of code, only to find out that I'd need to write my own workaround until these bugs are removed. (I have left comments in the code around sections that perform these functions.)

I also decided to implement my own pop-down menu rather than use the Athena "SimpleMenu" widget, simply as a matter of implementation choice. I wasn't happy with the appearance or behavior of the Athena widget, and wrote my own.

I'm not unhappy with my decision to use Athena. Aside from these problems, the basics of Athena worked fine. The interface is clean and simple, and I was able to complete the project without the great attention to detail that Xlib programming requires. However, a high-priority future goal is to move XBUILD to a "professional" widget package such as Motif.

1.3 Implementation Notes

These are some of the goals that I strived to achieve as I wrote the various pieces that make up XBUILD. I'll outline them here.

Provide a consistent user interface.

Based on many different studies of human-computer interfaces as well as books written on the subject (such as [1]), I have tried to conform to a consistent, well-designed user interface. This means paying attention to details such as which side the "Confirm" vs. "Cancel" buttons are on, providing consistent mouse commands, confirming decisions that might lose the user's data, and so forth. Some of my design comes from examination of operating systems such as Apple's Macintosh System 7 and Commodore-Amiga's Workbench[3].

Conform to standards.

This applies to both my choice of environment and my programming practices. I wanted to write clean, portable code that would be compatible with

[3]A perfect example of what *not* to do if you want to provide a clean, consistent interface: Commodore released the operating system years before they had any standards for developers. The result was hundreds of applications which functioned completely differently and had no way of exchanging data.

any system running in a similar environment and I wanted to steer clear from writing system-specific code and ignoring standard conventions of the interface (of which, in X, there are very few).

Provide a help system.

All forms generated with the XBUILD system are capable of having an integrated, non-intrusive help system defined by the form creator. I felt this was important in keeping with the theme of simplicity and ease-of-use I wanted in XBUILD.

Write modularly.

All relevant functions are kept together in one source file. The programs are split into as many as six source files (with appropriate header files, plus one header file for global definitions), each one a package of related functions and procedures. This makes it easy to find sections of code that need to be patched, modified or upgraded.

In addition, this makes it easy to add functionality to the program in the future. Adding more modules, or interchanging existing ones, is possible because of the way the program is designed.

Keep the program simple and easy to use.

Just as it says: Don't make the program so complicated that it can't be used by a beginner. A basic form requires no programming knowledge[4]. Later, as the user becomes comfortable with the program and how it works, he or she can take advantage of more advanced features.

Make sure it's as multi-purposed as possible.

XBUILD wasn't designed with a particular application in mind. It was built to be flexible[5], so that it could be applied to a variety of tasks. The only common thread is the nature of the tasks: They must involve the gathering of data into a form for some purpose.

[4]The script for the most basic form is two lines long: (1) fillform, (2) bsave *filename*. This script will allow the user to enter data into all of the available fields, and then append the data to a master database.

[5]XBUILD has been compared to HyperCard on the Macintosh. This is a valid comparison, but XBUILD wasn't written as an "X version of HyperCard". XBUILD was meant to be generic, but not omniscient.

Chapter 2

Using XBuild

XBuild is the first of the three programs that make up the complete package. XBuild allows you to create your forms, scripts, help panes and screens, and to save them all into a data file that will be used by XRead (the run-time form displayer).

2.1 Starting XBuild

We'll assume that you've already installed XBuild in a local directory, and have placed the XBuild resource file into the location where your implementation of X looks for such things.

Starting XBuild is as simple as typing:

```
% xbuild
```

You can also specify a form name right away, if you want to begin working on an old form you've already built:

```
% xbuild -form myform.xbd
```

You'll see the XBuild main screen (and your old form, if you used the `-form` option) appear.

2.2 Examining the XBuild screen

Take a moment to note the way that the different areas are laid out. XBuild attempts to keep your tools and functions grouped into related areas so that

you can easily find what you're looking for. The sections following this one discuss each object and mode in detail, and how they can are used to create your form.

2.2.1 The Menu Strip

The top-most section is the menu strip. This area contains buttons that let you enter different sections of the program such as the screen editor or script editor.

The File button

This button lets you perform various functions, such as saving the current form, loading a new form, clearing the form area, and so forth. Pressing on the "File" button brings up a drop-down menu which you can exit simply by moving the mouse into and then out of the menu. To select one of the options on the menu, simply point to the option you want with the mouse, and click on it.

The menu contains the following options:

New Clears the current form from memory *without saving it first.* If you have made any changes to the current form without saving them, XBuild will caution you not to proceed before saving the form.

Open Loads an existing form into memory. Any changes in the current form *will be lost.* If you have any unsaved changes in the current form, XBuild will caution you not to proceed before saving the form.

Save Saves the current form from memory into a data file. You will be asked for the filename if you haven't already given your form one. If your form already has a name, then XBuild will quietly replace the old version with your current one, and not ask you for a different name. (XBuild will also *not* warn you that you are about to save over an existing file, because it assumes that you mean to save the changes to your current form, and know that it already exists.)

Save As.. Saves the current form from memory into a data file, but always asks you for a new filename first. If you enter a filename that already exists, XBuild will ask you if you wish to replace it.

Quit Exits XBuild. If you have any unsaved changes, XBuild will ask you if you really want to quit, since doing so will cause all of your changes to disappear.

The Script button

This button lets you edit the script for the current form. When you click on this button, an editor window pops up and shows you the current script (new forms have an empty script). Script editing is discussed in Section 2.7, and extensively in Chapter 3.

The Execute button

When you're almost done with your form, you should use this button to see how your form will look when it's passed through the reader (XRead). This mode also syntax-checks your script, and attempts to verify that the field names you reference are valid. It's always a good idea to pass a form through this mode *before you run it through XRead*, as the error messages generated by this mode are more extensive (and therefore, more helpful in determining where your error is). This mode is discussed further in Section 2.9.

The Screen Editor button

This button lets you edit the various information screens you may wish to create (or have already created) for the current form. When you click on this button, a screen editor window pops up and allows you to select a screen to edit, create, or delete. Screens are discussed in Section 2.8.

The Grid Modifier Buttons

Both the "Grid +" and "Grid -" buttons modify the automatic "snap-to" grid in the drawing area. When you first start XBuild, this grid is set to increments of ten pixels. This means that whenever you add, move or resize an object, it will only move in ten-pixel increments when you move the mouse. This is meant to help you align objects easily. If you wish to enlarge the grid size, click on "Grid +". If you wish to decrease the grid size, click on "Grid -". A grid size of 1 is equivalent to having no grid. The maximum value the grid can have is 100 pixels.

2.2.2 The Label Area

The area below the menu strip is the label area, which will tell you the name of the current form you're working on. If you're starting a new form, the label area will read "File is: Untitled". This means that you have never saved the current form, and have never given it a filename.

2.2.3 The Toolbox

The area below the label area is the toolbox, which allows you to select what kind of object (a field or a box, for example) you wish to add or change in the drawing area. You can select a tool simply by pointing to it with the mouse and clicking on it. The tool you select will become highlighted.

One tool, the "Delete" tool, is not really a tool at all. It is a mode that lets you delete any object in the drawing area simply by clicking twice on the object.

These tools are available in the toolbox:

Field The Field tool lets you add fields to your form. Fields are perhaps the most important object you'll be adding to your form. They allow you to gather data from the user, and also act as your "variables" when you write scripts. We'll discuss fields later on in Section 2.3.

Text The Text tool lets you add any text you wish to your form. You might want to add some text that tells the user what to enter in each field, for example, or you might simply want to put a title on your form. Text boxes are discussed in Section 2.4.

Box The Box tool lets you add a box to your form. You could use a box to group related fields for the user, or to call attention to important fields on the form. Boxes are discussed in Section 2.5.

Line The Line tool lets you add a line to your form. This could be used for separating different sections of your form, or for pointing to related fields. Lines are discussed in Section 2.5.

Delete The Delete tool lets you delete any object on your form. Deleting objects is discussed in Section 2.6.

2.2.4 The Drawing Area

The largest area is your drawing area. This is where you'll be creating your form (or editing an existing one). You select a tool from the toolbox, and use the mouse to create and edit objects of that tool type in the drawing area. The amount of drawing space you have in the drawing area is exactly what will be available in the reader tool (XRead), so you can lay out the various components of your form exactly as you'd like them to be displayed to the user.

2.3 Fields

Fields are perhaps the most important objects that you will be adding to your form. Fields are objects that allow users to enter information in a pre-determined format. This information is called a field's *value*. Fields are identified by their name, which you use in scripts to retrieve their values. Fields also have other attributes which can be set in the *field dialogue box*, which we'll discuss shortly.

2.3.1 Adding a Field

You add a field to a form by following these steps:

- Click on the "Field" tool in the toolbox.
- Click and *hold down* the left mouse button on the location in the drawing area where you want the *top left* corner of the field to go.
- Drag the mouse down and to the right.
- When the field is the size you want, release the mouse button.

The field will be given a default name (usually "Field_x", where x is a sequential number) which you can change. The field has also been given some default attributes, all of which can be edited through the field dialogue box.

2.3.2 Editing a Field's Attributes

Fields have several attributes associated with them. They are:

Name This is what identifies a field uniquely. You reference a field in a script by its name, and you can choose to have the field's name displayed or hidden (discussed below). Choose a field name that's short and easy to remember, and use underscores instead of spaces. The maximum length a field name can be is 30 characters.

Allow This attribute describes what kind of characters can be entered into the field. It can have four values:

Alphas Upper and lower-case characters.

Numerics Numbers (0 through 9).

Specials Any character not included in the Alphas or Numerics category, *except* the space character.

Spaces Blank characters (as typed with the space bar).

You can set all or none of the categories, or any combination of the four. This is only one way to restrict what kind of information is entered into the field. (The other way is through a preset, discussed next).

Preset Specifies exactly what kind of input a field is to take. A preset specifies how a field takes input in two ways: The *format* of the data and the *characters allowed* in the data. XBuild comes with four pre-defined presets called "Name" (which allows a string of Alpha characters and spaces, no preset format); "Social Security" (which allows numbers and dashes only, in the format "___ __ ____"); "Telephone" (which allows numbers, dashes, and parentheses, in the format "(___) ___ ____"); "State" (which allows only Alphas, in the format "__"). Presets can also have a post-conversion option, which is a conversion that is applied after the data is entered, verified, and forced into the preset's format. The types of conversion available are:

C Convert to all upper-case.

R No conversion.

L Convert to all lower-case.

N Convert the first letter of each word to upper-case (Name conversion).

Of the standard XBuild presets, "State" uses "C" conversion, and "Name" uses "N" conversion. The others use "R", or no conversion.

Label Indicates whether or not the field should show its name as a label when the form is run. Labels appear on the left side of a field, so if this attribute is turned on, make sure you've left room for the label on the form.

Lookup Indicates that the field should have a value entered when the form is first run. The values you can retrieve are:

UID The user's current login ID.

Date The current date.

Length The length of the field, in characters. This value also affects loading text from files into a field. Under no circumstances can you have more characters in a field than this attribute specifies, so be careful when specifying this limit.

Protect Whether or not this field can have its value altered. For a field that has a "Lookup" attribute specified, this attribute might be set so that the user can't modify the looked-up value.

Scrollbars Indicates whether or not this field will present "scrollbars" to the user which allow the actual text inside the field to be larger than the actual size of the field. The text does not wrap around; instead, the user pans through the text by clicking on a slider bar with the mouse. Scrollbar fields are useful when you want to read a large file into a field for display but you don't know how large that file might be. The scrollbars allow the user to use the field as a "viewport" on the larger amount of text within. Scrollbars appear on the bottom of the field (for horizontal movement) and on the left of the field (for vertical movement). It is recommended that you set the "Protect" attribute on when you use the scrollbar option, and only use a scrollbar field for displaying data, not accepting it. Make sure to set your "Length"

attribute suitably high so that it can accept large amounts of text to display.

Each field also has a "help panel" associated with it. This is a small blurb of text that will help the user if he or she is having any trouble figuring out how to proceed. This blurb appears directly below the field it belongs to when the user clicks on a button in XRead marked "Toggle Help".

To edit a field's attributes, click on the "Field" tool in the toolbox, then move the mouse cursor over the field you wish to edit. Double-click the left mouse button, and the field dialogue box will pop up. The various attributes that were discussed above are all represented in this box, and can be set or modified here. By default, when a new field is created, it has its length set to 80, label set to show, lookup set to none, protect set off, and it allows alphas and spaces.

To modify the Allow, Label, Lookup, Protect and Scrollbar attributes, simply click on the buttons that correspond to the value you wish. For example, to turn a field's protection on, you'd click on the box marked "On" next to the "Protect" attribute.

To modify a field's name or length, move the cursor over the box next to the "Field Name:" or "Length" attribute, press "Control-K" (hold down the control key and press "K") on the keyboard to erase what's currently there, and then enter a new value. (Names cannot exceed thirty characters. There is no restriction on field length, except for machine memory limitations.)

To modify a field's preset value, click once with the left mouse button on the button next to the "Use Preset:" attribute. You'll see a drop-down menu listing the available presets you can select. Move the mouse cursor over the preset you wish to use, and click once with the left mouse button. If you decide not to use one of the presets, move the mouse cursor over the "No Preset" option, and click on that. You can change what presets are available, and how they behave. This is discussed in Appendix A.

You should also make sure that the field has a help panel defined. To edit a field's help panel, click on the button marked "Edit Help Box..". A small editor panel pops up. Type your help message in, and click on the "Done" button to continue. If you change a previously-written help blurb, and decide that you'd rather not make any changes, click on "Revert to Last Saved", and you'll see the help text that was there before you made any changes.

When you're through editing the field's attributes, click on the "Okay" button to complete your changes. If you decide that you don't like the changes you made, click "Cancel", and they will be discarded (*including* any changes you made to the help panel).

2.3.3 Changing a Field's Size and Location

You change a field's location by following these steps:

- Move the mouse cursor over the field you wish to move.
- Click and hold down the *middle* mouse button.
- Drag the field to the new location.
- Release the middle mouse button.

You change a field's size by following these steps:

- Move the mouse cursor over the field you wish to resize.
- Click and hold down the *right* mouse button.
- Drag the mouse. You'll see an outline following your mouse pointer which shows you the new size and shape of your field.
- When you're happy with the size, release the right mouse button.

2.4 Text

Text boxes allow you to add text to your form. For example, you might want to put a title on your form, or label all the different fields with something other than their name. In XBuild, a text box looks like a field, but when you pass the cursor over it you can type into it and add whatever text you like.

2.4.1 Adding Text Boxes

To add a text box to your form, follow these steps:

- Click on the "Text" tool in the toolbox.
- Click and *hold down* the left mouse button on the location in the drawing area where you want the *top left* corner of the text box to go.
- Drag the mouse down and to the right.
- When the text box is the size you want, release the mouse button.

When you pass the mouse cursor over any text box, a small character called a "caret" or "insertion point" appears at the place where text will be inserted if you type. You can only type into a text box when the mouse cursor is positioned over it. (Basically, whatever text box has the insertion point showing is the one you'll be able to add text to. Only one text box can be "active" at a time.)

2.4.2 Changing a Text Box's Size and Location

You change a text box's location by following these steps:

- Move the mouse cursor over text box you wish to move.
- Click and hold down the *middle* mouse button.
- Drag the box to the new location.
- Release the middle mouse button.

You change a text box's size by following these steps:

- Move the mouse cursor over the text box you wish to resize.
- Click and hold down the *right* mouse button.
- Drag the mouse. You'll see an outline following your mouse pointer which shows you the new size and shape of your text box.
- When you're happy with the size, release the right mouse button.

Something important to note: If you should resize the text box smaller than it was previously, you may lose some text that you typed in. A text box *cannot* hold more text than is visible on the screen. So, for example, if you had a three-line text box filled with text and you were to resize it vertically down to one line, you would lose the entire text of lines two and three.

2.5 Lines and Boxes

You can add lines and boxes to your form to provide a sense of grouping. If you have a form with sections of related fields and you want to group them together visually, you might surround them with a box; or, if you have two completely different sections of your form (say, a "query" section and a "results" section), you might draw a line between the two to call attention to the fact that they are different sections.

2.5.1 Adding a Line

To add a line to your form, follow these steps:

- Click on the "Line" tool in the toolbox.
- Click and *hold down* the left mouse button on the location in the drawing area where you want the first endpoint of the line to go.
- Drag the mouse. You'll see the line following your mouse pointer.
- When the line is the size you want, release the mouse button.

2.5.2 Resizing Lines

You change a line's size by following these steps:

- Move the mouse cursor over the line you wish to resize.
- Click and hold down the *right* mouse button.
- Drag the mouse. You'll see an outline following your mouse pointer which shows you the new size of your line.
- When you're happy with the size, release the right mouse button.

2.5.3 Adding a Box

To add a box to your form, follow these steps:

- Click on the "Box" tool in the toolbox.
- Click and *hold down* the left mouse button on the location in the drawing area where you want the *top left* corner of the box to go.
- Drag the mouse down and to the right.
- When the box is the size you want, release the mouse button.

2.5.4 Resizing Boxes

You change a box's size by following these steps:

- Move the mouse cursor over the border of the box you wish to resize.
- Click and hold down the *right* mouse button.
- Drag the mouse. You'll see an outline following your mouse pointer which shows you the new size and shape of your box.
- When you're happy with the size, release the right mouse button.

2.5.5 Moving Lines and Boxes

To move a line or a box on your form, follow these steps:

- Click and hold the *middle* mouse button on the border of a box, or directly on a line you wish to move.
- Drag the mouse. You'll see the object following your pointer.
- When the line or box is in the position you want, release the mouse button.

2.6 Deleting Objects

Any object can be deleted by following these steps:

- Click on the "Delete" tool in the toolbox.
- Move the mouse pointer over the object you wish to delete. (For boxes, make sure you move directly above of one of the borders.)
- Click the left mouse button twice quickly.

The object will be removed from the form. Be careful: You cannot undo a deletion, so be careful when you delete an object.

2.7 The Script Editor

A script is a list of commands which tell a form what to do. A form with no script does nothing, and therefore is of little use. This section will show you how to enter the most basic script. For complete information on scripting, see Chapter 3.

2.7.1 Entering the Editor

To get to the script editor, click on the button marked "Script" in the menu area. A script editor screen will pop up. You enter commands into the text editor region (with an insert point character showing). When you're done entering your script, click the "Save" button to go back to editing your form. If you don't want to save the changes you've made, click on the "Cancel" button.

2.7.2 A Basic Script

Every form must have a script to make it actually do something. If you're just interested in gathering information into a database for report generation, the script in this section will let you get by. But for anything more advanced, take some time and read Chapter 3.

The basic format of a line in a script is:

```
label command parameters
```

where *label* is a line label (for your personal use – you can't reference lines by their label, only by their number), *command* is a script command, and *parameters* are the various options that each command takes.

So, for example, the first line in our basic script might look like this:

```
0 confirm "Fill in the form?" 1 3
```

The label on this line is "0", the command on this line is "confirm", and the parameters are everything after the command. (To find out what the "confirm" command does, see Appendix B.)

The two commands you must know for basic scripting are "fillform" and "bsave". "fillform" activates the form for the user, and allows him or her to enter information into your fields. When the user is done, he or she clicks on a button marked "Done". Your script then resumes on the next line.

"bsave" puts data from your form into a datafile. The format of the datafile is special and unique to XBUILD, so you can't use it for exporting information to other programs. (There's a different script command for doing that, though.) All of the fields on your form have their values saved into the datafile; you don't have to be concerned about telling XBuild which fields to save. The set of values for all of your fields is called a *record*, and "bsave" knows to save the entire record in one chunk. If the datafile already contains other records, then the new record is appended to the end.

"fillform" has no parameters, and sits on a line by itself, i.e.:

```
0 fillform
```

"bsave", in its most simple form, has one parameter: the filename that you wish to use for the database. So, "bsave" would look like this in a script:

```
1 bsave /home/ta/mnc/data/database.txt
```

Putting it all together, our basic script would look like this:

```
0 fillform
1 bsave /home/ta/mnc/data/database.txt
```

Note: Line numbering begins at zero. The first line of a script is always considered to be "line zero", *not* "line one". It doesn't matter what a line's

label is, all that matters is the line's position in the script. It is suggested that you always use line numbers as labels for your own reference. Just remember to start your numbering from the top, from number zero.

The first line of our simple script activates the form and lets the user enter data. The second line saves all the entered data into a master database file.

If you enter this script into the script editor as it appears here (changing the filename on the "bsave" line to something more appropriate for your application), it will allow your form to accept values for every field and will save them into a file for later retrieval (with XStat, discussed in Chapter 5).

It is strongly suggested that you learn a little more about scripting, at least enough to put up information screens that explain why you want the user to fill in the form in the first place. But with this script you've mastered the basics, and it works.

2.8 The Screen Editor

You may wish to add informative screens to your form that will tell the user what the form is about, how they go about filling in the fields, how to get help, and so forth. Your form should have a proper title screen appear when the user first starts the program, so that they know who made the form and why they should take the time to fill it out.

You'll need to learn a scripting command in order to use screens, because you need to tell the program exactly when to pop up which screen. The forms displayer can't do that automatically, so you have to tell it in your script. This command is discussed in Section 2.8.4.

Screens can be created, edited and deleted through the screen editor. To create a new screen, or to see the screens you've already created, click on the "Screen Editor" button in the menu area. You'll see the screen editor dialogue box pop up.

The screen editor is divided into two parts. The left-hand column lists the screen names you've created. The right-hand box displays the text in a particular screen, once you've selected one. When the dialogue box first pops up, no screen is selected, and the right-hand box is empty (and "dimmed", to indicate that it is not accepting input or displaying any valid information).

Click on "Done" to dismiss the screen editor dialogue box.

2.8.1 Adding Screens

To add a screen to your form, follow these steps:

- Click on the "Screen Editor" button in the menu area. The screen editor dialogue box pops up.
- Click on the "New" button in the dialogue box. The new screen's name will appear in the left-hand column inside a circular button. Screens are automatically named "Screen_x", where x is a sequential number. You can't change the name of a screen.
- Click on the new screen's button in the left-hand column. The right-hand box will turn from grey to solid, and you may begin typing your text into the box. Whatever text you enter into the box will be *exactly* what is presented to the user when you display this screen in XRead.
- When you're done adding text, you can click on the "Done" button to dismiss the editor, or the "New" button to add another screen.

2.8.2 Editing Screens

To edit one or more of the screens you've already created, follow these steps:

- Click on the "Screen Editor" button in the menu area. The screen editor dialogue box pops up.
- Click on the button of the screen you wish to edit (listed in the left-hand column of the dialogue box). The screen will appear in the right-hand box.
- Type in new text, or change existing text in the right-hand box. When you're done, you can click on the "Done" button to dismiss the editor, another screen's button to edit another screen, or the "New" button to create a new screen.

2.8.3 Deleting Screens

Deleting a screen is not an undo-able option, so be careful when you choose a screen to delete. To delete a screen you don't want to use anymore, follow these steps:

- Click on the "Screen Editor" button in the menu area. The screen editor dialogue box pops up.
- Click on the button of the screen you wish to delete (listed in the left-hand column of the dialogue box). The screen will appear in the right-hand box.
- Click on the "Delete" button. Your screen will be deleted, and the right-hand box will become empty.
- You can now delete another screen in the same manner, create a new screen, or click "Done" to dismiss the dialogue box.

2.8.4 Displaying Screens from the Reader

Now that you've created some screens, you need to indicate in the script where each one should be displayed to the user. The script command to do this is called "showscreen", and it has the following format:

```
showscreen screen_name
```

screen_name is the name of the screen you want to display. "showscreen" pops up the screen in a box and puts a button at the bottom of the box marked "Continue". The script is halted at this point until the user clicks on this button. When he or she does, the screen is popped back down, and the script continues on the next line.

Normally, you display a screen as the first action of a form, and therefore you need to put a "showscreen" command on the first line of your script. Your "line zero" in the script could read like this:

```
0 showscreen screen_0
```

In this example, `screen_0` is the name of the screen containing all of the introductory text.

What comes next is up to you. If we also wanted a screen to appear after the user has finished filling in the form, we could add another "showscreen" command after a "fillform" command. Referring to our basic example script in Section 2.7.2, we could add "showscreen" commands like this:

```
0 showscreen screen_0
1 fillform
2 showscreen screen_0
3 bsave /home/ta/mnc/data/database.txt
```

This script would show our first introductory screen as soon as the form was started up (called `screen_0` in this example). When the user clicks on the "Continue" button, the screen is popped down, and the user is allowed to fill in the fields on the form. When the user clicks the "Done" button, another screen is displayed, and again waits for the user to click "Continue". Then, the information is saved to a database file, and the form exits.

2.9 Trying Out your Form

When you are using XBuild to design your form, you spend most of your time in what is called the *edit mode*. This is the mode you're in whenever you are moving, adding or resizing objects, editing screens, or editing the script. In this mode, what you see on the screen is an almost-exact replica of what the user will see when he or she runs your form through the reader portion of the package, XRead. But it's not *exactly* what they will see. For example, text boxes won't have an outline around them like they do in XBuild at all times, and fields won't have their labels in the center.

In order to get a better idea of exactly what your form will look like when it's run, you can enter the "execute" mode. This mode will give you an exact replica of your form as it would be displayed by XRead.

Execute mode also checks your script for any syntax errors[1] or semantic errors[2], and warns you if you have anything that might make your form not behave the way you want it to when it's passed through the reader. It will also warn you if you haven't written any script, because a form without a script won't do anything when run through XRead. It is *very* important to make sure that your script passes through execute mode without any errors.

To enter the execute mode, click on the button marked "Execute" in the menu area. You'll see some changes in the way your form is laid out and displayed. A new button will appear in the menu area which reads, "Return

[1] Errors like misspelling a command, or using a command that does not exist.

[2] Errors like referencing a screen name that does not exist, or a field name that is improperly spelled and can't be found.

to edit mode". You can click on this button when you're ready to go back to editing your form.

You can't make any changes to your form while you're in execute mode. You can, however, enter text into the various fields to get a sense of what it will look like in XRead. Simply point the mouse cursor over the field you wish to type in, and type some text.

If you have any errors in your script, you'll see a dialogue box which will display the type of error, and the line number that it occured on. Click the button marked "Dismiss" to continue, return to edit mode, and fix your script. When you think you've corrected your error, return to the execute mode again.

If you enter the execute mode and no dialogue box pops up, then your script is error-free and can be run through XRead.

Fields that have their look-up attribute specified will display "*lookup_value*" in the field area. This indicates that in XRead, these fields would have the proper lookup value already entered. You won't be allowed to type into fields that have their protection attribute set, and they will have a grey outline box instead of a black one.

Also notice that fields that have their label attribute turned on will show their names to the left of the field entry box. This is important to note, because you need to make sure that the field is positioned properly in the form to allow the label to be displayed. Check to make sure that no field's label is cut off by another field, or the edge of the form.

Using the execute mode will help you design forms that are pleasing to look at as well as use. Take advantage of this mode's true WYSIWYG[3] display to aid you when building a form.

[3]What You See Is What You Get.

Chapter 3

Scripting

Scripting is an important part of learning to use XBUILD, although if you aren't interested in learning more than the basics, you don't have to learn anything more than what was presented in the last chapter. This chapter aims to give the intermediate to advanced form builder the information they need to build forms that can do things more complex than simple information-gathering. Quick-reference tables which list the commands and their parameters start on page 57 (Appendix B).

3.1 The Basics of Scripting

A script is a list of actions which tell the form reader program (XRead) what to do. These actions are also called *commands*. There are twenty-four available scripting commands in the package which do things from saving and loading data to clearing the form or a field.

Some commands need additional information, called *parameters*, which provide more information about the command you wish to use. For example, the "showscreen" command has one parameter: The screen number you wish to display. The parameter for the "showscreen" command would be "screen_1", if you wanted to display screen_1. (Some commands have no parameters, and are used on a line by themselves).

Each command is put on a single line in the script, with parameters immediately after the command on the same line. Lines are numbered, beginning at *zero*, not one. The format of a single line looks like this:

```
label command parameters
```

where `command` is the command name and `parameters` are the unique set of parameters that the particular command takes. `label` is any word, number or character you wish to use to identify the line. Since line numbers are *not* displayed in the script editor, it is a good idea for you use a line number as the label so that you can refer to a line in the script correctly. (This is actually what the label area is intended for.)

Some commands, such as "goto", take a line number as their parameter. Make sure that you enter a correct line number based on the line's *position in the script*, and not the line's label. Lines are *never* referenced by label, only by their unique position. Remember that lines are numbered beginning at *zero*.

3.2 Script Commands

Commands are listed alphabetically in this section. The first line of each section lists the command and its proper usage in a box. A description of the command follows the usage, and any notes about restrictions on a command follow the descriptive paragraph.

3.2.1 addquit

```
addquit
```

Adds a quit button to the menu area in XRead. After this line in a script, the user can abort the program (and, therefore, the script) *at any time the form is active* simply by clicking on the "Quit" button, so it is imperative that you make sure data can't be destroyed if the user selects this button during a "fillform" command. This command would be useful in an endless retrieval loop, so that the user could stop reading data from a database whenever he or she so desired.

3.2.2 brestore

`brestore` { *field=value*} *filename*

Restores data previously saved in the XBUILD-1 format into the current form. The field names stored in the file are matched to the field names of the current form, and data is placed when a match occurs. If no match occurs (either because the form was changed or the datafile does not match the form), then the field data is discarded and no error is presented to the user.

If *field=value* is specified, then the record restored must make this expression true. If no record exists to satisfy this expression, no record is retrieved. If *field=value* is not specified, then the first record is retrieved.

field is NOT enclosed in hard braces. Field names cannot have spaces.

field=value has the same format and features as the "restore" command (Section 3.2.17).

3.2.3 bsave

`bsave` {*key=field*} *filename*

Saves all data from all fields into a file in the XBUILD-1 binary data format. This is a format only recognizable to the XBUILD suite of programs. *key=field* has the same format and features as the "save" command (Section 3.2.20).

Note that if you plan to use XStat on any data entered by your end-users, *you must save the data in this way!* XStat can only read data files saved in this format.

3.2.4 clearform

`clearform`

Clears all data from the current form. All field values will be blank.

3.2.5 clearfield

`clearfield` *field*

Clears the value of *field.*

3.2.6 confirm

`confirm` *mesg line1 line2*

Pops up a confirmation dialogue, with OK and CANCEL buttons, and message text *mesg*. *line1* and *line2* are script line numbers, which indicate the next line to execute if the user presses OK or CANCEL, respectively.

Notes:

- *mesg* **must be** enclosed in double quotes:

```
"This is a proper message."
```

3.2.7 copy

`copy` *file1 file2*

Copies *file1* to *file2*. If *file1* does not exist, or if *file2* cannot be created, an error will be reported.

3.2.8 desense

`desense` *field*

Makes the specified *field* unsensitive to user input. This means that the field will not be included in tabbing order (i.e., the user won't be able to use "tab" to select your field) and will not accept text input. Use to protect a field that you no longer wish to allow the user to change. You can re-sensitize a field with the "sense" command (Section 3.2.21).

Notes:

- This command *cannot* be used on a field that has its "scrollbar" attribute set, for logistic reasons.

3.2.9 exit

`exit`

Immediately exits the form and the reader program.

3.2.10 fillform

```
fillform
```

All gadgets and fields on the form "awaken", and the user is in control until the "Done" gadget is pressed.

3.2.11 goto

```
goto line
```

Branches execution of a script immediately to line *line*.

3.2.12 if

```
if field="value" line
```

Branches execution of a script to line *line*, if and only if the *field* equals the specified *value*. *value* **must** be contained within double-quotes, and can contain spaces.

3.2.13 loadfile

```
loadfile filename field
```

Loads the contents of an *entire* file specified by *filename* into the named *field*. Most commonly, *field* will be a scrollbar field to allow the user to scan through the file at their convenience. Make sure that *field* has a large enough length attribute to accomodate the size of the file, otherwise the field will display only as much of the file as it can accomodate, and truncate the rest. (The original file will not be modified.)

3.2.14 mail

```
mail userid subject file
```

Mails *file* to the user with userid *user*, with *subject* as the message's subject. If *file* does not exist, or *userid* is invalid, an error code will be returned by XRead from the mailer shell, and an error will be reported to the user.

3.2.15 pause

```
pause msg line
```

Creates a button on the top of XRead's screen, with *msg* for as the button's label text. The form is deactivated (i.e., no modifications can be made by the user). When the button is clicked, the script continues on line *line.*

This is useful for a retrieval system, after data is retrieved, to introduce a display screen and wait until the user has read the screen and is ready to move on.

3.2.16 remove

```
remove file
```

Removes *file*, quietly. No errors are reported.

3.2.17 restore

```
restore {field=value} filename formatstring length
```

Imports raw data from a file to the fields named in the *formatstring*, in the order and format also named in the *formatstring.* If *field=value* is specified, then import only the first record that makes this expression true.

field is NOT enclosed in hard braces in this use. Field names cannot have spaces.

value can be any string *without blanks.* Do not enclose this value in double-quotes, or the quotes will be considered part of the value. A space terminates the value string. *value* can also be one of these special words (which must appear between hard-brackets):

[UID] Value becomes the user's current login ID.

[VALUE] Value takes on the current value of the specified *field.*

formatstring must be in the same format as the "save" command uses (see section 3.2.20). Note that the special keywords allowed in the "save" formatstring (such as [ALL]) do not make sense in the "restore" formatstring,

and are not allowed. The backslash character (\) can be used to represent a newline in the file (mostly used when *length* is zero).

length is the length of a single record, in bytes, as was specified in the save command. *length* can be zero, which indicates that you wish to read from a free-form file (i.e., no fixed record length), in the format indicated in the *formatstring*. This is used mainly for importing data from other programs that are capable of generating straight ASCII files in a specific format.

For example, say you wish to import data into a form that has three fields: Name, Address, and Phone. Your file has this data, in the following format:

```
Joe Jones
123 Anystreet
512-123-1234
Jane Doe
912 Disk Drive
212-313-4414
```

Your "restore" command would look like this:

```
restore /usr/tmp/data.dat "[Name]\[Address]\[Phone]\" 0
```

Recall that "\" is a special format indicator which stands for a carriage-return or newline.

Note the "0" at the end. This specifies that you are importing data with no fixed record length. Also note that the format string ends with a delimiter (in this case, a carriage- return). **This is very important!** You *must* make sure to include the final delimiter so that XRead knows when to stop reading one record and start reading another.

This restore command restores the first record in the file. Using the key specifier allows you to pull out a specific entry from your data file. (For example, adding "`Phone=512-123-1234`" after the "restore" command would retrieve Joe Jones' entry.)

Notes:

- The file must agree with the format specified in *formatstring*, or extremely unreliable results may occur[1].

[1]The possibility for a core dump exists here, due to a quirk in the way the `strtok()` function works. The form designer must be careful when using the restore command

3.2.18 retrieve

`retrieve` *field register*

Copies the value of *register* into a *field.* If the specified register is empty, nothing is copied. Register values are specified by number (1 through 10 only). Store data into registers using the "store" command (Section 3.2.24).

3.2.19 run

`run` *cmd formatstring*

Runs a program and pass it the *formatstring* as a parameter list. *cmd* is the command to be run. The *formatstring* may contain field names, enclosed in hard braces. For example:

```
"-e [Field1] -f [Field2] -iconic"
```

would pass the value of Field1 to the "-e" flag, and the value of Field2 to the "-f" flag of the command to be run. "`-iconic`" would also be passed to the command.

3.2.20 save

`save` {*key=fieldname*} *filename formatstring length*

Saves field values as ASCII data into a file, in the format given by the *formatstring. key=fieldname* is an optional parameter which marks the indicated field as the "key" to the file. If a key field is given but the key value does not already exist in the file, the key is created and the data entered into the record access by that key. If the given key value exists in the data file, data is written to the file beginning at the offset where the data matches the key value. If no key is given, the data is simply appended to the file.

fieldname is not enclosed in brackets or quotes. Field names cannot contain any spaces. The word *key* must be present if you are specifying a key.

---

to make sure the file agrees with the formatstring. This limitation is one of the most important ones to be removed in version 2 of the XBuild package.

length is the length (in bytes) of one record, and cannot change once the file is created. This parameter **must** be specified, but can be zero. If the length is specified as zero, then "save" will simply append data in the format specified by *formatstring* to the end of the file, and not attempt to do any key or value matching. In fact, if length is zero then you are not allowed to specify a key, and an error will be generated if you try to do so.

Data is entered into the file based on the *formatstring*. A *formatstring* specifies names of fields separated by any ASCII characters the user wishes. Some examples:

```
"[Name];[Address];[field_3];"
"[Name] [Address] [Field]"
```

Note that field names within the *formatstring* **must** be enclosed in hard braces, therefore, hard braces are considered reserved characters and cannot appear elsewhere in the *formatstring*.

Notes:

- **Important:** If you allow the user to enter a character in a field which is also used as a separator in the *formatstring* (i.e., if you use ";" to separate fields in the *formatstring*, and you allow the user to type ";" into a field), you *will not be able to read this data with the "restore" command!* XRead will have NO idea where the end of user-entered data is and where the next field starts. You *can* use the "save" command to save this data, but *don't* try to use the "restore" command. If you need to save and restore data in this format for some reason, use the "bsave" and "brestore" commands and use XStat to generate files in the format you need; or, use both the "save" and "bsave"/"brestore" in the script.

- There are a few reserved words which can be used in the *formatstring* that do special things:

  [UID] Substitutes the userID for the current user.

  [ALL] Writes out all of the field values, separated by semicolons.

- A backslash (\) appearing in the *formatstring* is converted into a newline.

3.2.21 sense

```
sense field
```

Makes the specified *field* sensitive to user input. This means that the field will be included in tabbing order (i.e., the user can use "tab" to select your field) and will accept text input. Use to re-sensitize a field that has been desensitized either by the "desense" command (Section 3.2.8) or by having its protect attribute set (Section 2.3.2).

Notes:

- This command *cannot* be used on a field that has its "scrollbar" attribute set, for logistic reasons.

3.2.22 showscreen

```
showscreen name
```

Displays the screen called *name*. Screens are named automatically in the form builder.

3.2.23 sleep

```
sleep secs
```

Halts the execution of the program for *secs* seconds. Useful when you want to introduce a "timing catch-up" pause in your scripts. For example, suppose you want to use the "restore" command (Section 3.2.17) to read data from a file that will be created by a process you start with the "run" command (Section 3.2.19). There will be a time delay between the time the file is completely created and the time you want to read it, so you might have your script wait for 4 seconds between the "run" command and the "restore" command.

3.2.24 store

```
store field register
```

Stores the current value of a *field* into one of ten registers. The contents of the field can be any type, any length (the registers will expand or shrink to fit the length of the data). *register* is specified by number (1 through 10 only). This provides a way to use fields as variables internally while still retaining the original data in the field. You can move data from registers to fields using the "retrieve" command (Section 3.2.18).

3.3 An Example Script

In this section we will present an example script, and go through it line-by-line to get a feel of how scripts are put together. This script would be a good starting point for you to write your own scripts.

Let's go through the script. Refer to Figure 3.1 which shows the script as it would look in XBuild's script editor. This script is used with an information-gathering form that will be used by students to enter their address, phone number, and any comments about their coursework that they care to make. This information is saved to a master database file, keyed by the student's user ID. The form has three informative screens called "Screen_1" through "Screen_3".

```
0 brestore UserID=[UID] /data/iform.dat
1 showscreen screen_1
2 fillform
3 confirm "Do you want to save this information?" 4 7
4 bsave Key=UserID /data/iform.dat
5 showscreen screen_2
6 exit
7 showscreen screen_3
8 exit
```

Figure 3.1: The example script

- Line 0 attempts to find the database file located in the directory "/data" called "iform.dat". It also indicates that the restore should attempt to

find a record where the field "UserID" equals the user's current user ID ("[UID]" is a special value in the "brestore" command).

- Line 1 displays the screen called "screen_1". The script is halted here until the user clicks on the "Continue" button located at the bottom of the displayed screen's box. When the button is clicked, the screen is popped down, and the script continues on line 2.

- Line 2 activates the form and allows the user to enter information into the form. The script is once again halted here until the user clicks on the "Done" button in XRead's menu area. When the user is finished with the form and clicks on the "Done" button, the script continues on line 3.

- Line 3 pops up a dialogue box with the message, "Do you want to save this information?" and two buttons marked "Yes" and "No". The two numbers on this script line indicate which line of the script will be next, depending on what the user clicks on in the dialogue box. If the user clicks on the "Yes" button, the script will resume on line 4. If the user clicks on the "No" button, the script will resume on line 7.

- Line 4 saves all of the data that the user entered into the master database file located in the directory "/data" called "iform.dat". It also indicates that the "UserID" field should be used as a key, which means that it should look through the database file, and try to match the current value of the UserID field with the values of that field in the database. When it finds a match, it should *replace* that record with the current one. If no match exists it will simply append the current record to the end of the database file.

- Line 5 displays the screen called "screen_2". Again, the script is halted here until the user clicks on the "Continue" button. The script resumes on line 6.

- Line 6 exits the form and quits the program.

- Line 7 displays the screen called "screen_3". The script is halted here until the user clicks on the "Continue" button. The script resumes on line 8.

- Line 8 exits the form and quits the program. Note that a script does not have to have an exit line if it doesn't need to. In this script, the "exit" statement was needed on line 6 to prevent the script from continuing on into line 7, but the "exit" statement on this line is unnecessary.

Chapter 4

Using XRead

XRead is the form interpreter that the end-user will actually run when he or she is using the forms you have designed. This section details XRead's basic use and some possible installations.

4.1 Starting XRead

We'll assume that you've already installed XRead in a local directory, and have placed the XRead resource file in the location where your implementation of X looks for such things. In addition, you must have already created a form with XBuild. If you haven't, go back to Chapter 2 and create a form.

Starting XRead is as simple as typing:

```
% xread -form myform.xbd
```

where "myform.xbd" is the name of the form you've already created. If you don't specify a form with the `-form` option, XRead will look for a file called "default.form". If it can't find that file, it will report an error and exit.

You can also specify that the "help mode" should be forced on at all times. To do this, add the flag `-help` on the command line when you invoke XRead:

```
% xread -help -form myform.xbd
```

Note that this is probably *not* the way you'd want to set things up for end-users to use your form. Read section 4.4 for some suggestions on installation.

4.2 Examining the XRead screen

XRead looks similar to XBuild, except that there are only two areas: The menu area, which is the topmost area of the screen, and the form area, directly beneath the menu area. Your form is displayed in the form area, and there are two buttons in the menu area marked "Toggle Help" and "Done". There is also a message which says, "Help mode off." In the form area, one of the fields on the form has a box around it, which indicates the *currently active field* (i.e., the one that you will type into).

The "Toggle Help" button turns the display of help panels on and off[1]. When you first start XRead, help mode is turned off, and no help is displayed. When you click on the "Toggle Help" button, the message in the menu area changes to "Help mode on", and a help panel appears beneath the active field. Click on this button again to turn help mode off again. Help panels will follow the currently active field as you change from field to field. You can turn help on and off whenever you like.

(If you click on the "Toggle Help" button and no help appears, it's probably because the field has no help panel defined. If you created the form, go back into XBuild and make sure you've written help panels for all user-accessable fields[2].)

The "Done" button is used to indicate that you have finished making your changes to the form, and are ready to go on. Don't click on "Done" unless you are really done with the form. Exactly what happens when this button is pressed depends on the script for the form, and it may not include confirming that you really wanted to click on the "Done" button.

There are two other buttons which may appear during the course of using the form, depending on the form and the script for the form. The "Quit Program" button allows you to exit the form at any time. When you're completely done with the form, you can click this button to exit immediately.

The "Click to Continue" button may also appear. This button can appear when you are examining the contents of a form, but not making any changes. Click this button when you're done and want to move on. (The exact wording on this button, and its exact behavior, may be different depending on the

[1] Unless the `-help` flag was specified when XRead was started up. In this event, the "Toggle Help" button has no effect.

[2] If you didn't create the form, find the person who did and tell them.

form and its script.)

4.3 Navigating the Form

You can move the active field by pressing the TAB key or the RETURN key[3]. You always type into the active field only. If you can't tab to a particular field, that field is probably marked as "protected". This means that the field contains information that you aren't allowed to modify.

You may see this message in a dialogue box when you press the TAB key:

```
The data in this field is invalid.  Please try again.  To
turn on help, click "Toggle Help"
```

This means that you have entered something that doesn't agree with the field's format. For example, entering a letter in a field that asks for a zip code would be invalid. Entering three numbers in a field asking for a phone number would also be invalid. If you can't figure out why a field isn't letting you enter information, click the "Toggle Help" button, and you should get an informative help panel that details the kind of information the field is looking for. If you *still* can't figure out what to do, just click on "Done" and seek help elsewhere.

Depending on what form you're filling out, there are several different kinds of fields you'll see on the form. The most basic is an input field, which looks like a box with an "insertion point" marker (a small up-arrow) which indicates where you'll be typing. These types of fields allow you to type in text in either a free-form format or a specific format, depending on the type of information the field is set up for. (For example, a telephone field is looking for text in the form of a phone number: Typing `516-555-1212` would, therefore, be valid. Typing `Emu5` would not.)

Another type of field is called a "scrolling" field. A scrolling field looks like a regular field, except it has large, grey bars on the left and bottom sides. When the text inside a scrolling field exceeds the actual size of the field, these bars will begin to shrink in size. Then you can then use them to view different parts of the text by following these steps:

[3]The RETURN key is not used for moving to another field when you enter a *scrolling* field. The RETURN key is used to move down a line.

- Place the mouse cursor on top of the bar you wish to move. Use the bottom bar to scroll horizontally, and the left bar to scroll vertically.
- Click and *hold down* the middle mouse button.
- Drag the mouse up and down (if you clicked on the left bar), or left and right (if you clicked on the bottom bar). You'll see the text scroll with the mouse.
- When you're done scrolling, release the mouse button.

For horizontal scrollbars, you can also jump-scroll immediately to the right (the text will move to the left) by clicking once on the scrollbar with the left mouse button. To scroll to the left (the text will move to the right), click once on the scrollbar with the right mouse button. The amount of text that you jump is determined by the distance the mouse pointer is away from the left side of the scrollbar. The closer you position the mouse pointer to the left end of the bar, the smaller the jump will be.

For vertical scrollbars, the same method works, except that the left button scrolls you down (the text moves up), and the right button scrolls you up (the text moves down). The closer you are to the top of the scrollbar means the smaller the jump distance.

4.4 How to set XRead up

This section is mostly aimed towards the system administrator (or form builder) who would like to set XRead up to gather data with a particular form.

XRead takes a single parameter which tells it which form to run. The end-user shouldn't have to know what your form's data file is called, nor should he or she have to know the format of the XRead command to run the form. It is suggested that you set up a directory of forms, a directory of database files, and create aliases or shell scripts to run a particular form.

Here are some examples that show how XRead could be set up. We'll assume that you've installed XRead in the directory "/usr/local/bin" and that you've created a forms directory called "/usr/local/lib/xread/forms" and placed your forms there. For the sake of example, the form we'll set up

to run will be called "student.xbd", and we'll execute it with the command "userinfo".

4.4.1 Using Scripts

Place this script in your "/usr/local/bin" directory:

```
#!/bin/sh
/usr/local/bin/xread -form /usr/local/lib/xread/forms/student.xbd
```

Name this script "userinfo". Make sure the permissions are set so that the file is executable (`755` are decent permissions). Now, all your end-users have to do to run your form is type "userinfo", and XRead will automatically be loaded with the proper form. You can create as many script files as you wish to run XRead with different forms.

4.4.2 Using Aliases

This method might be a little harder, because it requires changing the `.login` or `.cshrc` of each of your end-users. If this isn't a problem, then it might be a better choice than using scripts because it avoids cluttering up a directory with additional executable files.

In the user's `.cshrc` file, enter a line that will create an alias like this:

```
alias userinfo "../reader/xread -form ../forms/iform.xbd"
```

Once again, the user can simply type "userinfo" to run XRead preloaded with your form.

Chapter 5

Using XStat

XStat is a utility to help you manage the databases you will generate as your forms get used. XStat has basic maintenance capabilities, a "global find" function, and a powerful reporting function.

5.1 Starting XStat

We'll assume that you've already installed XStat in a local directory, and have placed the XStat resource file in the location where your implementation of X looks for such things. In addition, you must have already created a form with XBuild, and collected some data into a file using the "bsave" command in the script.

If you haven't created a form yet, go back to Chapter 2 and create a form. If your existing form hasn't been saving data into a file using "bsave", XStat *cannot* work with your form's database. XStat can only read database files in the format generated by "bsave".

Starting XStat is as simple as typing:

```
% xstat -form myform.xbd
```

where "myform.xbd" is the name of the form you've already created. If you don't specify a form with the `-form` option, XStat will look for a file called "default.form". If it can't find that file, it will report an error and exit.

If the form you specify has no "bsave" line, XStat will report that it can't find a proper database file and exit.

You can force XStat to pull database information from a specific file by using the `-db` option:

```
% xstat -db mydb.dat -form myform.xbd
```

5.2 Examining the XStat screen

XStat, like XRead, is divided into two sections: The menu area, which is the topmost area, and the form area, directly beneath the menu area. The menu area contains buttons which allow you to perform various functions with the database file. These buttons are, from left to right:

Quit Exits the XStat program.

Stats Shows a small window with some basic statistics about the form, such as the filename, number of responses, and number of fields in the form.

Report Allows you to generate a custom report file from your database file. See section 5.5 for details on how to create a report.

Find Allows you to search for a string of text in any field of any record. See section 5.3 for detauls on how to perform searches.

Find Again Repeats the last "find" command.

Delete Deletes the record currently being displayed. Does not actually modify the database file until the "Quit" option is chosen (and even then, asks the user if he or she really wishes to modify the database file permanently.)

Next Moves to the next record in the file and displays it.

Previous Moves to the previous record in the file and displays it.

First Moves to the first record in the file and displays it.

Last Moves to the last record in the file and displays it.

"1 of *n*" Indicates the current position in the database file, where n is the total number of records.

5.3 Finding Records

To find a particular record in the database, follow these steps:

- Click on the button marked "Find". A dialogue box will pop up.
- Enter the text you wish to search for. The text can appear in any form in the field.
- Click on the "Find" button in the dialogue box.
- If the text you entered appears in a record, that record will immediately be displayed. If the text you entered does not appear in any record, a dialogue box will be displayed that says, "String not found." Click on the "Continue" button to continue. The form you were previously viewing will still be displayed.

You can repeat a search without having to enter the text again. Clicking on the "Find Again" button will repeat the last search you did without asking you to enter any text.

XStat always looks *forward* when performing a search. For example, if you were currently looking at record 6 of 10 records, and started a search, XStat would only search records 7 through 10 to find the string. XStat does *not* do "wrap around" searches. If XStat reports that it can't find a string, make sure you've begun your search at the *first* record to ensure that you're searching through all the records in the database. It is always a good idea to click the "First" button before initiating a new search.

5.4 Moving through the records

Clicking on the "Next" button in the menu area will move you forward one record. Clicking on the "Previous" button in the menu area will move you backward one record. When XStat reaches the end of the file, and you click on the "Next" button, XStat "wraps around" to the first record. Similarly, if you were at the first record of the file, and you clicked on "Previous", XStat would show you the last record of the file.

You can jump to the first record of the database by clicking on the "First" button. Similarly, you can jump to the last record of the database by clicking on the "Last" button.

5.5 Reporting

Reporting allows you to create standard ASCII files of the data you've collected in any format you wish. Reporting follows many of the same conventions that format strings do in the script language, but the mechanism is more interactive than that of the script.

To generate a report, click on the "Report" button in the menu area. You'll see the report generator dialogue box pop up. The large box on top is the area for the *report format string*, which is what tells XStat what to put in the report. The smaller box on the bottom of the dialogue box is where you should enter the filename of the report you wish to generate. Make sure to enter the full directory path if you don't want to create the file in the current directory. Clicking on "Generate Report" generates the report in the specified file, assuming a proper report format string and filename has been entered.

The report format string specifies what each record in the file will look like when it is written to the report file. You can enter any text you like. To include the value of a field, you enter the field name, enclosed in hard brackets.

For example, if you wanted to generate a mailing list from all of the people in your database, you might use a format string that looked like this:

```
[Name]
[Address]
[City], [State] [Zip]
```

This assumes that you have fields called "Name", "Address", "City", "State" and "Zip" in your form. Note that you could also have included any text in the format string you liked, such as this:

```
Name:  [Name]
Address:  [Address]
City:  [City]
State:  [State]
Zip:  [Zip]
```

Note the blank line at the end of the format string: This makes sure that a blank line will be inserted between every record in the report file.

Sample output from the first format string would look like this:

```
Marcus N. Cannava
1170 Genesee Street, A5-6
Rochester, NY 14611-1020

Phil White
123 Disk Drive
Rochester, NY 12313

Homer Simpson
123 Springfield Way
Springfield, PT 10102-3012
```

Sample output from the second format string would look like this:

```
Name: Marcus N. Cannava
Address: 1170 Genesee Street, A5-6
City: Rochester
State: NY
Zip: 14611-1020

Name: Phil White
Address: 123 Disk Drive
City: Rochester
State: NY
Zip: 12313

Name: Homer Simpson
Address: 123 Springfield Way
City: Springfield
State: PT
Zip: 10102-3012
```

5.6 Deleting Records

To delete a record from the database file, follow these steps:

- Display the record you wish to delete on the screen. You can do this by using the "Next" or "Previous" buttons, by using the "Find" command, or by using the "First" or "Last" buttons.
- When the record you want is the one displayed, click on the "Delete" button in the menu area. The record will disappear, and you will see the previous record displayed.

Note that this does *not* actually delete the record from the database file yet. When you select the "Quit" option, XStat will ask you if you really want to make these changes permanent. If you do, click on the "Save Changes" button. If you don't want to make your deletions permanent, click on the "Discard Changes" button.

Appendix A

Customizing XBuild and XRead

XBuild, XRead and XStat all have corresponding resource files that need to be installed in the resource directory of your local installation. Resource files tell the windowing system about various aspects of the program, such as window size, font, color, and so forth. Some of the resources listed in the resource files for the XBUILD package are used internally, and changing them would cause the program to look strange (at best), or completely stop working (at worst). *Never* change a resource file unless you have a backup copy of the unmodified file.

This section lists the resources that are intended to be changed to customize the way the package works.

A.1 Preset Configuration

You can configure XBuild & XRead with up to 4 preset field types. XBuild & XRead come configured with a Name field, a Social Security Number field, a Date field, and a Telephone field. You can access these presets with the resources:

```
*presetxName:
*presetxFormat:
*presetxAllow:
*presetxOptions:
```

where *x* is the number of the preset (1 through 4. *Never* modify preset zero). Simply add these resource names, with appropriate values as described below, to the XBuild **and XRead** resource files to modify a preset. (Note: XBuild and XRead *must* contain the same preset resources, or else strange behavior may result).

The "Name" resource specifies the name of the preset. You can use any string you like, up to 15 characters.

The "Format" resource specifies what format, if any, the string should take. For example, you can specify a phone number's format like this:

```
(__) ___-____
```

Then, whatever the user enters in the field, it will be forced into this format you specify. If there is an error in matching the input data to the format string (for example, the user didn't enter enough numbers to fulfill the format), the user will be informed and asked to re-do the field.

Use the "_" character in the format string to indicate where user-entered characters should go. Any other character is considered part of the format[1]

The "Allow" resource specifies what characters the user should be allowed to enter into the field. You do this by specifying ranges of ASCII codes, separated by commas. So, a valid "Allow" specification, which only lets lowercase and uppercase letters be typed by the user, would look like this:

```
65-90,97-122
```

Ranges can also simply be one number. For example, this line would allow A-Z, a-z, and the "Space" character.

```
65-90,32,97-122
```

The "Options" resource specifies what post-processing should happen to the string, after formatting has taken place. This is a single-character option, and can be one of R, C, N, or L:

R No changes.

C Capitalize all alpha characters.

[1]There is currently no way to make the "_" character itself a part of the format string — it is always substituted with a user's input character. This probably should be resolved in the next major release of XBuild.

N Capitalize as a "name" (i.e., first character of each word).

L Lowercase all alpha characters.

Here's an example entry, exactly as it would look in the XBuild & XRead resource files, to make a preset called "State". This preset takes the position of preset 4. It would only allow two characters, from A-Z or a-z, and capitalize them when done:

```
*preset4Name:          State
*preset4Format:        --
*preset4Allow:         65-90,97-122
*preset4Options:       C
```

A.2 Cosmetic Changes

You can change the font used by either XBuild or XRead by changing the font resource:

```
*font: -adobe-helvetica-bold-r-*-*-12-*-*-*-*-*-*-*
```

The font specified can be any valid, installed X font.

You can change the colors used by either XBuild or XRead by changing the color resources:

```
*Command.background:           lightblue
*Command.foreground:           black
*Box.background:               grey
*drawbox.background:           white
```

The "Command" resources modify the color of the menu and dialogue buttons. The "Box" resource modifies the background color of the dialogue and label boxes. The "Drawbox" resource modifies the color of the main drawing areas.

You can change the shape of command buttons by changing the Command resource:

```
*Command.shapeStyle:           ShapeOval
```

The valid values are "ShapeOval" (the default) or "ShapeRectangle".

A.3 Language

Any of the resources that end in "`.label:`" are language resources. These can be changed for the purpose of translating the package into a different language, or for changing the wording of the various buttons and fields.

A.4 Warnings

Don't modify any of the resources that haven't been mentioned here. They control the delicate geometry and spacing of various pieces of the XBUILD package. Also, don't attempt to run any of the three programs without installing their respective resource files first.

Appendix B

Script Commands

Command	Args	Description
showscreen	name	Display a text window, keyed by the window's name as defined by the form's author.
fillform	*none*	All gadgets in the form awaken, and the user is in control until the "Done" gadget is pressed.
clearform	*none*	Clear all data from the current form.
clearfield	field	Clear all data from the specified field.
sense	field	Make the specified field "sensitive" to user input. *(Not valid on scrollbar fields.)*
desense	field	Make the specified field "insensitive" to user input. The field will lose its border and input point marker. *(Not valid on scrollbar fields.)*

Table B.1: Commands to manage the form, fields & screens.

Command	Args	Description
store	field rnum	Store the value of *field* in register *rnum*.
retrieve	field rnum	Retrieve the value in register *rnum* into *field*.

Table B.2: Commands to deal with registers.

Command	Args	Description
confirm	mesg n m	Pop up a confirmation dialogue, with "Ok" and "Cancel" buttons, and message text *mesg*. *n* and *m* are script line numbers for "Ok" and "Cancel", respectively.
exit	*none*	Exits the form/program immediately.
goto	n	Go to line *n* in a script.
pause	msg lineno	Waits for the user to click the pause button, which will be labeled with *msg*. Script continues at line *lineno*.
addquit	*none*	Makes a "quit" button appear which allows the user to exit the form at any time.
if	field="value" line	Branch to line *line*, if the *field* equals the specified *value*.
sleep	secs	Makes the program wait for *secs* seconds to pass.

Table B.3: Commands to manage flow of control.

Command	Args	Description
save	*key=field* filename formatstring length	Save the data given by *formatstring* into *filename* with record length *length*. *key=field* is optional.
restore	*field=value* filename formatstring length	Restore data from *filename* by the format in *formatstring* with *length* record length. *field=value* is optional.
bsave	*key=field* filename	Saves data from all fields in an internal format
brestore	*field=value* filename	Restore data from a file saved with *bsave*.
loadfile	filename field	Load the file *filename* into field *field*. Typically, *field* is a protected, scrollbar field.

Table B.4: Commands to handle I/O.

Command	Args	Description
mail	userid subject file	Mail *file* to user *userid* with subject *subject*.
run	cmd formatstring	Run *cmd* with parameters *formatstring*.
copy	file1 file2	Copy *file1* to *file2*.
remove	file	Remove *file* quietly. No errors are reported.

Table B.5: Utility commands.

Appendix C

Error Messages

These messages are generated by the script interpreter in both XBuild and XRead to try to help you to pinpoint any mistakes made in your scripts. Most contain line numbers so that you know exactly where the error ocurred.

Error: The form "*filename*" does not exist.

Displayed when XBuild is given the `-form` option, but can't find the specified form.

Script error: *command* command on line *line* is malformed.

Displayed when a line in your script has the correct command, but an invalid number of parameters for that command.

Script line *line*: Syntax error in restore command.

Displayed when the "restore" command is used with an improper number of parameters, or with a malformed key.

Script error: Centers are used improperly in "if" command on line *line*

Displayed when the "value" parameter of an "if" command is either not centerd properly (or at all), or centerd in an irregular manner.

Script error at line *line*: No such command.

Displayed when a command is used that isn't a valid script command.

Script error: File *filename* does not exist, can't copy it!

Displayed when the "copy" command can't find the file specified to copy.

Script error: Register value *value* too high: store skipped. Script error: Register value *value* invalid: retrieve skipped.

Displayed if the register number in either the "store" or "retrieve" commands is specified as greater than 10, or less than 1.

Warning: Script is empty. Forms will not function without a script.

Displayed if the execute mode is entered, but the script is blank. Forms without scripts can't do anything, so this warning message is displayed to remind the form builder that a script needs to be written.

Script error: There's no screen called *screen.*

Displayed if a screen name used in the "showscreen" does not exist.

Script error: No such field *field* in *command.* Script error: No such field *field*: *command* skipped. Script error: No such field *field* for *command* key. Script error: No such field *field* in *command* format.

These errors (and others like them) are displayed when an invalid field name is discovered in a formatstring, a key identifier, or a value identifier of one of these commands. They all indicate that the field name is either misspelled, or does not exist at all.

Script error: Line references on Confirm statement are invalid.

This error indicates that one (or both) of the reference numbers at the end of the confirm statement refers to a line number that does not exist. Check the line counting, and be sure that the confirm statement references actual line numbers, and not line labels. (Remember that the first line is line *zero*, not line one.)

Appendix D

Troubleshooting

This chapter lists some problems that might come up while using the three programs in the XBUILD package.

Problem When I start up XRead, the program briefly shows my form, and then exits immediately.

Solution Your form has no script associated with it. Forms without scripts can't do anything. Go back to XBuild, and write a script.

Problem I double-clicked on a field in XBuild, but the attribute dialogue box didn't pop up.

Solution You need to have the field tool selected in the toolbox in order to edit a field's attributes. Click on the "field" tool in the toolbox, and then double-click on the field you wish to edit.

Problem In XRead, whenever I type into one of the fields on my form, it just beeps and doesn't let me type anything.

Solution You specified the length of the field to be zero. Go back to XBuild, edit the field's attributes, and set the length to something greater than zero.

Problem In XRead, whenever I type and whenever I press the TAB or RETURN keys, it just beeps.

Solution The mouse pointer is pointing inside a help box. Move the mouse pointer outside of the help box (but still inside the form).

Problem In XRead, whenever I try to tab out of a field, it keeps telling me that the data I entered is invalid, no matter what I type in.

Solution The "allow" attribute for that field has nothing set. A field should always allow some type of characters to be entered. Go back to XBuild, edit the field's attributes, and set the "allow" attribute to allow some set of characters. Or, change the preset from "No Preset" to one of the four available presets.

Problem In XStat, whenever I try to find a string I know exists in the database file, it says "String not found."

Solution Remember that XStat always searches *forward* from the current record, and does not "wrap-around" at the end. Make sure that you begin your searches from the first record.

Problem I retrieved an entry that I made with XRead a few weeks ago, and the lookup fields reflected the information that was *saved*, not the current values that should have been looked up.

Solution That is correct. The priority is always given to restored data from a database file. When you use the "bsave" and "brestore" commands to save and restore data, you don't have any choice over what gets saved, so lookup field values get saved to the database file, and retrieved when you retrieve the form. However, if you really need to keep lookup values fresh, use "save" and "restore" to save the fields that aren't lookup fields. That way, you won't have to worry about old data getting restored on top of the current lookup values.

Problem My form has two fields. One of them starts up inactive, the other active. I use the "desense" and "sense" commands in the script to deactivate the active field and activate the inactive field. But when I tried doing this, *both* fields stayed inactive!

Solution The order in which you "sense" and "desense" matters a great deal. The rule of thumb to follow is: *Never allow all of the fields on a form to become inactive at any time.* When you used "desense" to make the active field inactive, you suddenly had a situation where the only fields on your form were inactive fields, and XRead had no "current" field to work with. There is no way out of this situation, except to rewrite the script so that this does not happen.

Appendix E

Future Enhancements

This is a "wish list" of features that I would like to incorporate into the package at a later date. The program as it stands at this writing fulfills the baseline specifications that I laid out when I started designing the program, but I feel that it could come much further and eventually develop into a professional, polished package.

E.1 Contacting the Author

If you're using XBUILD, and have any comments, questions, or enhancements you've made, please contact me. I'd be happy to incorporate the changes into the next release of the software.

As of this writing, I can be reached at:

E-mail:
`mnc@cs.rit.edu`

Snail-mail:
Marc Cannava
7 Vieckis Drive
Nashua, NH 03062

E.2 The Enhancement List

E.2.1 Small Details

- The RETURN key should be used as a keyboard shortcut for the "Okay" button on dialogue boxes. A restriction in Athena prevented this from happening in this release, but in the future, this should be a feature.
- Shift-TAB should move *back* a field in XRead.
- More lookup settings should be available in the field attribute box. Currently, only UID and date are available.

E.2.2 Large Details

- *Motif! Motif! Motif!* Move the program over to a "professional" widget set (such as Motif) and get away from the Athena oddities.
- Resizing should be handled more gracefully than I do it now. Since an X program is mostly at the mercy of the window manager, a resize could happen at any time. Currently, the program doesn't deal with this properly: Widgets lose their proportions, the coordinates aren't valid any more, and so forth. Some new section of code would need to deal with this problem.
- Add interrupts to the scripting language so that an event can trigger a script line directly. For example, add the ability to tie in the clicking of a button to the execution of a specific line in the script. This may lead to eventually having separate scripts for different objects on the form (but this is getting dangerously close to what HyperCard on the Macintosh already does).

E.2.3 Feature Additions

- XStat should have more analysis functions. Add functions to produce mathematical computations on numerical fields, possibly plot graphs, and allow a report based on "How many people answered this field with *x*?".

- More field types should be added. Currently, there are standard fields and scrolling fields. Pop-down fields, lists, buttons, and radio buttons should be added.

Bibliography

[1] Apple Computer, Inc. *Apple Human Interface Guidelines: The Apple Desktop Interface.* Addison Wesley, 1988.

[2] Mark Cappel and Shalini Chatterjee. Goodbye open look, hello motif. *SunWorld*, 6(5):23–30, May 1993.